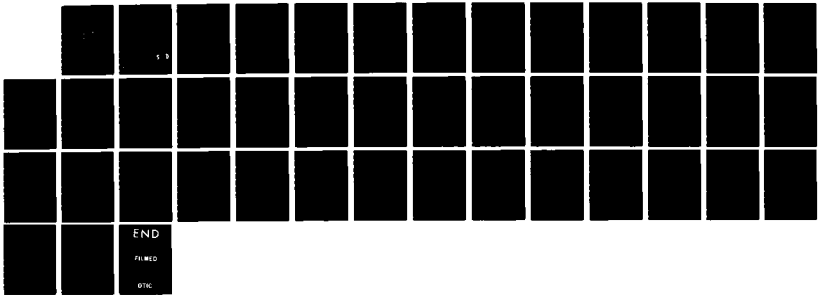
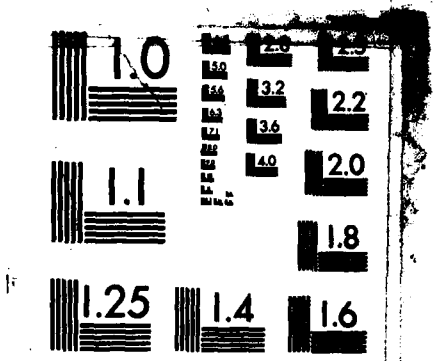
AD-A157 656 DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER ENGINEERING J A STANKOVIC 1/1

UNCLASSIFIED 15 FEB 85 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RESEARCH AND DEVELOPMENT TECHNICAL REPORT

CECOM

Decentralized Control of Scheduling in Distributed Systems

John A. Stankovic
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Final Report for the period Dec. 15, 1981 - Feb. 15, 1985

Distribution Statement

Approved for public release;
distribution unlimited

Prepared for
Center for Communications Systems (C. Graff)

CECOM
U S ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

Table of Contents

1.0 Introduction

A distributed processing system is defined as a collection of processor-memory pairs (hosts) that are physically and logically interconnected, with decentralized system-wide control of all resources, for the cooperative execution of application programs. Such systems may be dedicated to a single application or may implement a general purpose computing facility. By decentralized system-wide control is meant that there exists distributed resources in the system, that there is decentralized control of these resources (i.e., there is no single, central host "in charge", nor is there a central state table), that there is system-wide cooperation between independent hosts which results in a single unified system. By system-wide cooperation is meant that the algorithms of the system operate for the "good of the whole" and not for a particular host. For systems meeting this restrictive definition of distributed processing, it is hypothesized that their reliability, extensibility, and performance will be better than what is generally available today. In this report the term distributed processing refers to this very specific type of highly integrated distributed system.

The major objective of this research project was to develop and compare decentralized scheduling algorithms for distributed processing systems. The decentralized algorithms were to have replicated logic for reliability, yet cooperate to provide a system-wide sharing of resources. Using the algorithms as case studies, we would attempt to understand some fundamental issues regarding the performance and operation of decentralized control algorithms. Initially we implemented three simple algorithms to give us quick experience with decentralized control algorithms and to serve as a baseline

for comparison. If the simple algorithms work, why bother with more complicated ones? The results obtained from these three simple algorithms are interesting and are reported in section 2.1. We next implemented a sophisticated algorithm based on Bayesian decision theory with interesting results. These results are summarized in section 2.2. In parallel with developing the Bayesian decision theory approach, we also began to develop a general model for distributed scheduling based on bidding. Each instantiation of the model is actually a new algorithm. The results of this work are reported in section 2.3. We also studied the bidding approach for environments where tasks to be scheduled have deadlines. See section 2.4 for a description of this work. Another parallel effort was made to develop a decentralized scheduling algorithm based on stochastic learning automata. The results of this effort are described in section 2.5. Taken all together, we have developed a significant number of decentralized scheduling algorithms and have developed a good understanding of their operation. Overall conclusions of our efforts are presented in section 3.0.

2.0. Main Text

2.1. Three Simple Algorithms

A decentralized controlled job scheduling algorithm is one algorithm composed of n physically distributed entities, $\{e_1, e_2, \ldots e_n\}$. Each of the entities is considered a local controller. Each of these local controllers runs asynchronously and concurrently with the others, continually making decisions over time. Each entity e_i makes decisions based on a system-wide objective function, rather than on a local one. Each e_i makes decisions on an equal basis with the other entities (there is no master entity, even for a short period of time!). It is intended that the job scheduling algorithm adapts to the changing busyness of the various hosts in the system. In this section of the report, we further constrain job scheduling to operate with absolutely no a priori knowledge about the job. Its function is to assign jobs to hosts to provide a gross level of load balancing which in turn improves response time. Once a job is activated it is considered a process and is further scheduled by a process scheduler. A process scheduler can sometimes dynamically acquire information about a running process and therefore can be more sophisticated in its scheduling.

The environment in which the job scheduling entities are running is stochastic in two ways. First, the observations entities make about the state of the system are uncertain (they are estimate). Second, once a decision is made (e.g., move a job to host i), future random forces (e.g., a burst of jobs arrive at host i) that are independent of the control decision can occur.

Since job scheduling is an operating system function, any algorithm implementing job scheduling must run quickly. This is an extremely important aspect of the algorithms and makes many potential solutions unsuitable. For example, modeling the system by a mathematical program and solving it on-line is out of the question.

The execution costs involved in running a decentralized controlled job scheduling algorithm include the cost of running the algorithm itself, the cost of transmitting update information, and the cost of moving jobs. The primary goal of the algorithm is to minimize response time with minimum job movement. The secondary goal is to balance the load. All of these costs and goals are addressed with our simulations. We will not provide the details of our simulation model here. See [5].

We have developed 3 algorithms. They are as follows.

Algorithm 1. For moderate loads in the system, each entity compares its own busyness to its observation (estimate) of the busyness of the least busy host. Note that the host thought to be least busy is itself an estimate. The difference between the busyness of these two hosts is then compared to a bias. If the difference is less than the bias, then no job is moved, else, one job is moved to the least busy host. Jobs are not moved to oneself.

Algorithm 2. Each entity compares its own busyness to its observation (estimate) of the busyness of every other host. All differences less than or equal to bias1 imply no jobs are moved to those hosts. If the difference is greater than bias1 but less than bias2 then one job is moved there. If the difference is greater than or equal to bias2 then two jobs are moved there. In no case are more than (y)(z) jobs moved

at one time from a host, where y = fraction of jobs permitted to be moved, and z = number of jobs currently at this host. If there is more demand for jobs than an entity is permitted to move, it satisfies the demand in a pre-determined fixed order.

Algorithm 3. This algorithm performs in the same way as algorithm 1 except when an entity, e_i, sends a job to host k at time t, it records this fact. Then for time delta t, called a window, this entity will not send any more work to host k. If at any time during the window period, entity e_i calculates that host k is least busy then no job is moved during such an activation. Of course, during the window, jobs may be sent to other hosts if they are observed as least busy by greater than the bias (same bias as in algorithm 1).

One prime motivation behind these three algorithms is that they are all very simple and inexpensive to run, necessary conditions for job scheduling algorithms which have no a priori knowledge about jobs. In algorithm 1 the relative busyness between host i and the least busy host (plus a bias) is used to determine if a job should move. This is about the simplest algorithm we can devise, but one might suspect that it had stability problems. Since algorithm 1 only moves jobs to the least busy host we felt that a better algorithm might be to spread the work around, i.e., move some work to all the lightly loaded hosts from the heavily loaded ones. Hence algorithm 2 was devised. Finally, we were worried that jobs in transit to a lightly loaded host were not taken into account possibly producing instabilities. For example, if (1) host 1 was very busy, (2) host 5 was least busy, (3) host 1 were activated every 2 seconds, and (4) it took 16 seconds for jobs to reach host 5, then host 1 could conceivable send at last 8 jobs to host 5

before the first one was received. Other hosts could be doing the same thing. This could result in an unstable situation. Algorithm 3 was designed to avoid such problems in as simple a way as possible. More sophisticated techniques to deal with stability are possible, but they require the retention of more past data.

While we performed many simulations and compared these results to analytical models, in this report we concentrate on the conclusions rather than all the details of the simulations.

For light loads, algorithms 1 and 3 are about equal, improving the no network response time by about 50% at a cost of moving an average of between 11 and 14.8 percent of their jobs, respectively.

At moderate loads, algorithms 1 and 3 improve the no network response times by approximately 49%. Furthermore, for moderate loads algorithm 3 is superior to algorithm 1 because it achieves approximately the same performance (response time and load balancing) with less cost (job movement). Although such costs are already figured into response times we consider algorithm 3 better than algorithm 1 because we wish to avoid unnecessary job movement.

Overall, algorithm 2 performs even better than algorithms 1 and 3 but had a tendency to move too many jobs if not tuned well. The tuning is also more complicated because of the three biases, but as for algorithms 1 and 3 the chosen biases seem to be fairly robust over the various arrival rates and system conditions. This implies that the algorithm can retain its simplicity and operate over a wide range of arrival rates. A modification would be to dynamically adapt the parameters (biases) of the algorithm paying the associated cost. Such a modification could be the basis for a

future study. For moderate loads the response time of algorithm 2 was 61% better than the no network case, and approximately 24% better than algorithms 1 and 3. We, therefore recommend algorithm 2 but cutoffs, such as those found in algorithms 1 and 3 should be added to algorithm 2.

While startling results were not obtained, nor expected, the simulations did provide a number of insights. We now summarize these ideas.

It seems that even very simple decentralized control algorithms for job scheduling have a number of internal parameters that can significantly affect stability and response time. In our simulations it was possible to choose values for those parameters that worked well over the reasonably different arrival rates, subnet delays and scheduling intervals tested. In general, such parameters should be adaptive to insure stability and performance over a more diverse set of conditions.

The goal of our job scheduling algorithms is to provide a gross level of load balancing to insure adequate work for each host. If hosts are all very busy we found it to be imperative that scheduling is turned off under such conditions. Another potential cutoff involves avoiding the movement of large jobs because of their large delays in the subnet. This is one piece of a priori information that is easy to obtain.

In general, increased delays in the subnet increase system response time. However, we found that there is a flat portion to the response time curve where such an effect is felt only if the delay in the final queue (where a job actually receives service), plus the time needed to move is greater than the delay the job would experience without moving, or by moving faster. In other words, why hurry to the new host when you are just going to wait there too.

Increased delays also degrade the quality of the state information hosts have about each other. It would be interesting to better determine the effect of "out of date" information. Additional simulations were run to provide some insight into the effect of poorer and poorer information.

In some tests, we delayed state information by an extra 2, 3, 5, 7, and 10 seconds. The scheduling interval was 2 seconds. Hence, in the worst case, the scheduler was using out of date information from an immediate neighbor that was more than 10 seconds old. Algorithm 2 and 3 each show equivalence classes of response times (Table 1). In algorithm 2 the degradation in response time occurs when the data gets between 5 and 7 seconds old. Delays of less than 5 seconds result in about the same response time of roughly 19-21 sec, while delays between 7 and 10 seconds also have the same effect on response time but produce response times of approximately 26.5 sec. Algorithm 3 follows the same pattern but the break between equivalence classes occurs between 7 and 10 seconds. Such significant degradation in response time implies the need to take such poor information into account. The Bayesian decision theory technique reported in section 2.2 is such a technique.

When implementing a decentralized job scheduling algorithm an important question is how often to invoke the scheduler. Invoking the scheduler too often increases cost but a more important factor is the increase in non-beneficial job movement. While this result is interesting by itself, it has further implications. For example, as the cost of moving jobs increases, it becomes more important to move less jobs; one method is by slowing down the job scheduler. In some systems the overhead costs of the job scheduling algorithm could also play a major role, but in most of our simulations we

'able 1
elayed State Information

ALGORITHM		DELAYS 2 sec	3 sec	5 sec	7 sec	10 sec
2	Response T.	19.6	21.3	19.1	26.7	26.4
	% of Movement	41.2	53.7	40.5	85.6	77.1
3	Response T.	27.0	-	26.9	27.2	44.1
	% of Movement	33.4	-	47.4	59.4	75.5

Cost of Scheduler - 150 ms
Scheduling Interval - 2 sec.

kept this overhead small (50 ms). Increased costs of this overhead might occur due to different costs of context switches on different machines and operating systems. Table 2 shows the effect on Algorithms 1 and 3 of increased overhead. For our runs, the extra time required for the scheduler, by itself, would add 0.37 sec or 0.74 sec to response time for the 150 ms and 250 ms runs, respectively. But the overall effect is much more significant in a negative way (e.g., for algorithm 1 the actual degradation in response time is from 17.1 sec for the 50 ms case to 26.3 sec for the 150 ms case). Hence, it seems quite important to keep the cost of the algorithm low. Further, algorithm 1, because of the high percentage of movement, should be re-tuned for the higher scheduling costs. This adds credence to our observation that tuning needs to be adaptive and that algorithm 1 is not quite as good as the others because it is more prone to instabilities.

)le 2
ieduling Overheads

ALGORITHM		OVERHEADS 50 ms	150 ms	250 ms
1	Response T.	17.1±1.3	26.1	28.1
	% of Movement	35±2.6	87.4	95.2
3	Response T.	22.4±.7	27.0	27.3
	% of Movement	27±1.7	33.4	38.4

heduling Interval - 2 sec
lay per packet - 250 ms

2 Bayesian Decision Theory

The general model for Bayesian decision theory contains five in-edients and a set of maximizing actions. The ingredients are:

1. The set of available actions, $A=\{a_1, a_2, \ldots\}$.
2. The set of the states of nature which can occur $\theta=\{\theta_1, \theta_2, \ldots,\}$ and th probability distribution on the states of nature, $P(\theta)$.
3. The utility function $\mu(\theta, A)$, which contains the consequences of each combination of action and state of nature.
4. A set of possible observations $Z=\{z_1, z_2, \ldots,\}$ and the likelihood distribution $P(Z|\theta)$.
5. The choice criterion i to maximize expected utility.

Using the above ingredients a simple set of calculations is performed) produce a set of maximizing actions. The set of maximizing actions is a st of what action to take for each possible observation. Hence, once iese calculations are performed, a controller needs only to perform a table

aranteed, thus avoiding potentially unnecessary communication. This ap-
oach, however, would require time to check the node-surplus information to
termine potential bidders and could prevent bidding by nodes with surplus
the available information was inaccurate.

Bidding. When bidding in response to a request for bids, the bidder
rst estimates that its response will reach the requestor before the
sponse deadline. It proceeds with further actions only if the time of
sponse plus the transmit time for the response is less that the indicated
adline.

One a node decides to respond, it first computes ART, the estimated ar-
val time of the task if, indeed, it is awarded the task. Computation of
T takes into account the following: (1) the fact that bids at the re-
esting node are evaluated after the response deadline; (2) the average
lay in evaluating bids (estimated to be one half the bidding period); and
) the estimated time for the task to arrive at the bidder's node.

Whether the bidder can execute the new task is determined by the second
mponent of the bid, SARTD, which is the surplus at the bidder's node be-
een ART and the task deadline D. The surplus information takes into
count

(1) future instances of periodic tasks (to prevent jeopardizing
aranteed tasks);

(2) processing time for tasks that may arrive as a result of previous
ds (to ensure that nodes requesting bids are aware of other bids by a node
d minimize the probability of a node of being awarded tasks with conflict-
g requirements or being awarded too many tasks, creating an unstable
tuation); and

Request for Bids. For a task that cannot be locally guaranteed, the bidder broadcasts to all nodes a request-for-bid message containing the task's computation time C, deadline D, and size S; the time T at which the message is being sent; and a deadline for responses R. R is the time after which the requesting process will examine the bids to choose the best bidder. R should be such that after R there is sufficient time (1) for the requesting process to evaluate the bids, (2) for the task to reach the best bidder node, (3) for the next bidder to guarantee and schedule the task, and (4) once scheduled, for the task to complete computations and meet its deadline, Thus,

$$R=D-(P+E+W+C)$$

where P is the period of the task that evaluates bids and hence the maximum waiting time before bids are recognized; E is the estimate of the average time taken for the task to reach the best bidder; and W is a window representing the estimated time after arrival that the task might begin computation. Both E and W adapt to the load on the communication network as well as to the surplus of the receiving node.

If R is insufficient for the requests to reach other nodes and for the nodes to send their bids, the bidder then resorts to focused addressing. In this case, FP, the adaptive parameter used in focused addressing, is adjusted to augment the chances of finding a node with surplus. If a node with surplus still cannot be found, the task cannot be guaranteed.

A possible improvement over the above scheme for requesting bids, one in which RFBs are broadcast to all nodes, would be to send RFBs only to nodes whose estimated surplus matches the requirements of the task to be

communication delays will be estimated, based on a linear relationship between message length and communication delay. By utilizing the latest known delay, this computation can adapt to changing system loads.

In our scheme we assume that the clocks on different nodes are synchronized. The only effect of asynchrony in the clocks will be that the estimates, for example, of task arrival times and communication delays, will be inaccurate. When tasks are independent, a node can guarantee a task solely on local information, and asynchrony in the clocks will not affect such tasks.

We now describe the different phases of the bidding process. The bidder first checks whether it can send the task to another node via focused addressing, which utilizes surplus information to reduce overheads in the bidding process. If focused addressing can determine that a particular node has a surplus significantly greater than the new task's computation time and, thus, a high probability of guaranteeing it, the new task can be sent directly to that node without bidding.

We propose to do this in the following manner: Estimate the arrival time AT of the task at the selected node. If the estimated surplus of that node, between AT and the deadline D of the task, is greater than the computation time C of the task by FP percent, the task is sent to the node. The receiving node, as stated earlier, used the guarantee routine to check whether it can guarantee the arriving task. FP is an adaptive parameter used in focusing addressing.

If there is no node with a significant amount of surplus, bidding is invoked. In this case, the main functions of the bidder are sending out a request for bids for a task that cannot be guaranteed locally, evaluating bids, and responding to the request for bids from other nodes.

Interaction between schedulers on different nodes is based on a bidding scheme: the scheduler with a task that needs to be scheduled on that node itself, requests bids from nodes with surplus processing power. The bidder component of a scheduler utilizes the knowledge contained in each scheduler regarding the estimated surplus information passed between nodes in bids or is explicitly requested from other nodes.

Nodes making bids do not reserve resources needed to execute the task for which they are bidding. When a task arrives at a node, as the result of a bid being accepted, it is checked again to determine if it can be guaranteed. The node may be unable to guarantee the task if its surplus has changed since it sent the bid. This occurrence can be due to local task arrivals and to the arrival of tasks as a result of previous bids. One solution to this problem is for nodes to reserve resources, specifically CPU time slots, for the task for which they are bidding. We have not adopted this solution because of the poor resource utilization that it is likely to entail: a node may bid for more tasks than it will be awarded and there may be multiple bids for a task.

During the bidding process, it is necessary to consider communication delays. Communication delays depend on the pairs of processes involved, for instance, on the distance separating them and on the communication from other nodes in the system to the two nodes. We estimate the time in following way: every communication will be time-stamped by the sending node. The receiving node will then be able to compute the delay due to that communication by subtracting the time-stamp from the time of receipt. Subsequent

This research was directed at developing task-scheduling software for loosely-coupled systems that achieves flexibility through dynamic scheduling of tasks in a distributed, adaptive manner. We have developed

(1) a locally executed guarantee algorithm for periodic and non-periodic tasks, which determines whether or not a task can be guaranteed to meet its real-time requirements;

(2) a network-wide bidding algorithm suited to real-time constraints;

(3) criteria for preempting an executing task so that it still meets its deadline; and

(4) schemes for including different types of overheads, such as scheduling and communication.

The algorithm was developed with task independence assumed, and then extended to consider precedence constraints. The only resource requirements explicitly taken into account is CPU time. Many extensions are being considered and they are prime candidates for a follow on contract. Our scheme for distributed dynamic scheduling requires one scheduler per node. The schedulers interact to determine assignment for a newly arriving task. Associated with a node in a distributed system is a set, possibly null, of periodic tasks guaranteed to execute on that node. We assume that the characteristics of periodic tasks are known in advance and that such tasks must meet their deadlines. The existence of enough processing power at a node for all periodic tasks to meet their deadlines is verified at system initialization time. In addition to the periodic tasks, we allow for the arrival of nonperiodic tasks at any time and attempt to guarantee these tasks dynamically, in the presence of periodic tasks and on a network-wide basis.

proper set of parameters and their relative weights for the scheduling algorithm of a given system. We believe that good monitoring facilities coupled with an adaptive algorithm such as presented here can provide a form of "expert" operating system where choosing proper weights is feasible. Further, even today's uniprocessor operating systems typically have hundreds of parameters whose values are tuned for a given installation. While there is no science for choosing the values, experience has enabled effective choice of these parameters. We feel the same will be true of the parameters in an adaptive, distributed load balancing algorithm such as the one proposed here.

2.4 Real-Time Environment

Many tasks, such as those found in nuclear power and process control applications, are inherently distributed and have severe real-time constraints. Because their execution deadlines must be met, these tasks are said to have hard real-time constraints. Scheduling to meet their deadlines remains a major challenge in distributed system design. Although recent advances in software, hardware, and communication technology should permit more flexibility in designing these systems, most current research on scheduling tasks with hard real-time constraints has been restricted to multiprocessing systems. It is, therefore, inappropriate for distributed systems. In addition, many of the proposed algorithms, which assume that all tasks and their characteristics are known in advance, are designed for static scheduling.

The algorithm makes use of a modified McCulloch Pitts neuron calculation both to determine feasible candidates for movement, as well as to respond to requests for bids. This has been described in previous reports, so it is not repeated here.

To date, a simulation model has been implemented to test the variations of the algorithm under any number of system conditions. Simulation results can be compared to simple analytical models and baseline simulations, as we did, or where, for example, 100% accurate data is used instead of delayed information. Our current simulation results indicate that (under the conditions tested), (1) bidding is quite effective even with simple parameters, (2) the use of cluster and distributed group information in bidding is only marginally worthwhile (but its importance is expected to increase as the percentage of processes exhibiting cluster and group characteristics increase), and (3) transmitting bids a distance which is a function of the system load is effective. Of course, with such a complicated simulation model the simulation results have to be considered preliminary until a much wider set of conditions and combination of parameters are tested. Such tests are prime candidates for a follow on effort.

In summary, the advantages of our approach are that it is easily extensible to include any parameter deemed important for decision making, that the relative weights and functions that describe the importance of various parameters can be modified, that it attempts to load balance in nearest hosts first, that it includes resource, clustering and distribution requirements, and that tuning for a given network, application load and policy is possible. The main disadvantage is that it may be difficult to choose the

simultaneously, unshared or shared sequentially between domains and processes. For a process to execute, all segments must be local to the current site of execution of the process. A process may move around the network during its execution history, but whenever it moves all segments required for execution must be collected at the new local site. (We intend to allow remote access in future versions of our algorithm). We also include the notion of clusters and distributed groups.

A cluster is defined to be a set of processes that communicate frequently or in large volume, or both, in such a way that it is beneficial to schedule them on the same host. A cluster is is static if the collection of processes comprising the cluster is independent of time, otherwise it is dynamic.

A distributed group is as collection of processes that communicate with each other but would benefit from the inherent parallelism of being placed on separate hosts (all other things being equal). These benefits might accrue because the processes communicate via asynchronous send and receives or because the processes when separated don't have to contend for a shared resource such as a disk controller. One can think of a distributed group as the opposite of a cluster.

The bidding algorithm is replicated at each host of the network and is invoked periodically. The major functions of the algorithm are:

1. deciding on whether to transmit requests for bids,
2. transmitting bid requests,
3. processing requests for bids from other sites,
4. transmitting processes, and
5. dispatching local processes.

of the system parameters. The system parameters characterize different environments, different conditions in the same environment, and different policies.

4. The ability to adapt the set of sites which will receive the request for bids so as to be able to handle large networks and reduce the cost of bidding. The adaptation is a function of the load on a host, its collection of neighboring hosts, and possibly previous responses to bids.

The bidding function of our model attempts to match processes to processors based on many factors including process resource requirements, special resource needs, process priority, precedence constraints, the need for clustering and distributed groups (opposit of a cluster), specific features of heterogeneous hosts, and various other process and network characteristics. We use the framework of a McCulloch Pitts neuron to describe this portion of the model.

The evaluation of our algorithm was done by simulation. The results were compared to baselines and analytical models and indicate that bidding is effective using simple parameters when the system under study is modeled in a simple way, but when more realistic features are added, such as distributed groups, then more state information is required. We also found that adapting the distance a bid travels is a useful feature in improving performance and a necessity for large networks.

In this work a process is defined as an execution sequence through domains, where a domain is the process collection of segments (objects) at any point in time. A segment is defined as a logical unit of information with identical access characteristics. There are code, data, and environment segments. Depending on their characteristics, segments can be shared

model for distributed scheduling in highly cooperative computer networks. The model is based on bidding and serves as a means for developing sophisticated scheduling algorithms. The model can be considered a general template and a specific instance of the model is a distributed scheduling algorithm based on bidding. The advantage of our general model (and approach) include:

1. The ability to consider initial task allocation (where a task is a scheduler entity that has not yet begun execution), process scheduling (where a process is a task in execution), or both.
2. The ability to account for different "process" models, e.g.,
    1. all processes are independent and all resources required by a process must be local to that process,
    2. object based operating systems where all objects required by a process must be local to that process,
    3. object based operating systems where objects required by a process may be local or remote from the current site of the process,
    4. processes which communicate frequently or in high volume (clusters),
    5. processes which interact but would benefit from executing on different processors (distributed groups), and
    6. processes which communicate via different IPC mechanisms such as send and wait, or send and don't wait.
3. The ability to be tailored to a specific environment via the choice

values for the period of state information update and the period of recalculating the maximizing actions. Finally, we then performed a significant number of tests to determine the effect of various arrival rates and various delays in the subnet.

The most interesting result is that for moderate loads the improvement seen in the DBDT heuristic grows as the delay in the subnet grows. The reason is that the DBDT algorithm is equipped to identify the fact that the quality of its state information is degrading and uses that in making its decision.

In summary, several specific observations of our simulations include: that once tuned the algorithm works in a stable manner over a wide variety of conditions and over a large number of runs, that the probability distributions describing the true states of nature and the likelihoods of an observation converge to values representing the level of observability of the system, that the loss of monitor nodes does not cause major problems, that the heuristic works will under light loads but simpler approaches would be just as good, that the need for the heuristic increased both for moderate loads and for a decrease in the accuracy of state information available, and several thresholds included in the heuristic improve the operation of the algorithm.

2.3 A Distributed Scheduling Model Based on Bidding

Bidding algorithms have been accepted as a possible heuristic for cooperation among distributed components. Scheduling in highly cooperative distributed systems is an application area ideally suited to bidding although, to date, it has not been effectively evaluated. We have proposed a

Stage 2 is a completely dynamic BDT simulation, dynamically updating the a priori probabilities, maximizing actions, state information and performing job movement.

Under light loads there is a 50% improvement in response time of the Bayesian decision theory heuristic over th no network case. BDT performs similar to an analytical fractional assignment algorithm. The fractional assignment response time is optimal, is very optimistic and does not account for any costs. Using knowledge about the state of the network, our BDT heuristic performs better than knowing the statistical distributions of loads which is necessary to calculate the optimal fractional assignments. BDT was found to perform within 25% of a lower bound baseline which uses perfect information. This difference can be considered as the cost of inaccurate information.

For moderate loads there is again a 50% improvement in response time over for the BDT heuristic over the no network case. Here, though, BDT does considerable better than the fractional assignment. This is a clear indication that passing reasonably current tate information provides improved performance.

The dynamic Bayesian decision theory simulations has a number of important parameters including (i) the scheduling interval, (ii) the bias, (iii) the period of state information update, (iv) the period of recalculating the maximizing actions, (v) the arrival rates, and (vi) the delays in the subnet. Testing a large range of value for each parameter and in combinations with each other is prohibitive. To reduce the problem we chose a scheduling interval of 8 seconds and bias = 2 as derived from the Stage 1 simulation. Then, tuning the DBDT simulation consisted of finding good

its particular action which is often difficult if not impossible to obtain. Feedback occurs implicitly through the changing states of nature and the updating of the probability distributions. For example, as state information progresses through the network it affects the observations of the different controllers and thereby affecting their control decisions. The quality of the coordination and the resulting performance of the decentralized control algorithm is measured and tuned by simulation. In practice the tuning would be done on a real system.

A practical consideration for job scheduling algorithms comes into play for very lightly loaded and very heavily loaded systems. In both instances it is not beneficial to move jobs, therefore, jobs are not moved by a host if it observes a very lightly or very heavily loaded system. Very lightly loaded is defined as each host has less than 4 jobs. A very heavily loaded system is defined as each host had more than 20 jobs. All other situations are considered moderate loads.

The simulations of the BDT heuristic proceeded in two stages. Stage 1 is static in the sense that no updates of the a priori probabilities and maximizing actions are performed. It is dynamic in the sense of dynamically updating state information and performing job movement. The stage 1 simulations provided evidence that the heuristic could operate effectively and thereby merit further testing. Further, stage 1 simulations model the situation of losing all the monitor nodes, i.e., upon losing the ability to dynamically update the maximizing actions, the BDT scheduling heuristic would switch to maximizing actions modeled in stage 1. The results from stage 1 also serve as a point of comparison to the completely dynamic BDT simulation (stage 2).

look-up to determine what action to take given that it has made a particular observation.

The heuristic for applying this general model to decentralized control algorithms is to model each of the n job scheduling controllers as a Bayesian decision maker with the states of nature and observations being defined network-wide. Periodically, state information is passed between the n controllers so that each controller has a reasonable approximation about the state of the network. The period of update is an important parameter both for cost and for the relative accuracy of the data involved. In addition, special monitor nodes of the network act to dynamically adjust the probability distributions $P(\theta)$ and $P_i\ (Z|\theta)$, $i = 1, 2, \ldots n$ by gathering statistics to recalculate the maximizing actions, and to downline load these maximizing actions to each of the n scheduling controllers. This dynamic re-calculation of maximizing actions is a centralized aspect to our heuristic and can also be considered a form of cooperation because each controller informs a centralized component of its true state and its observed state at time t. This information is then used to alter the individual probability distributions needed by each of the controllers. This centralized aspect of our heuristic is permitted because (a) it is a convenient way to calculate the true state (note that it is a true state that has occurred in the past) and, (b) if the centralized monitor node crashes the decentralized controllers can continue to function using the last or a default set of maximizing actions until a backup monitor is activated.

Coordination between the decision makers is accomplished in an implicit manner by passing state information around the network. In other words, there is no direct feedback to a controller of the system-wide goodness of

(3) processing time needed for nonperiodic tasks that may arrive locally in the future (to minimize the probability of a task arriving as a result of a bid not being guaranteed due to the arrival of a local task with similar real-time requirements).

Bid processing. Bids are processed by the node that originally sent the request for bids. A bid processor queues all bids until the response deadline R. Once the deadline passes, the bid processor computes the task's estimated time of arrival at each bidder's node. For each bidder it estimates SETAD, the surplus between ETA and D, assuming the same rate of surplus indicated by the bidder. SETAD is computed using the following formula:

$$SETAD = SARTD * (D-ETA)/(D-ART)$$

The bid processor then chooses the one with the greatest SETAD as the best bidder.

To the best bidder, the task is sent. The identity of the second best bidder, if any, is also communicated to the best bidder. So that bidders will know the response to their bids and can purge unnecesaary information, the bid processor intimates to all but the best bidder that their bids were not accepted. A possible alternative, which avoids the traffic resulting from the responses, is for bidders to time-out after a predetermined interval.

One final note about the information sent on bids: A node utilizes it to track surpluses in other nodes, and this surplus information is used for focused addressing and for sending RFBs to nodes with high surplus. In responding to an RFB for a task, a bidder sends the estimated arrival time

an the estimates surplus between the arrival time and the task's deadline. If a node does not respond to an RFB, it is assumed that it does not have sufficient surplus. (In reality, a node may not have sent its bid because it estimated that its bid will not arrive before the deadline for responses.) Since information received via bids is bound to be fragmented, a node may explicitly request surplus information from other nodes within a specific time interval. Whichever scheme is followed for obtaining surplus information, due to the nature of system activities, it will often be outdated by the time it is used. Hence, such information should be viewed as estimates that will be updated when nodes bid.

Response to task award. Once a task is awarded to a node, the awardee node treats it as a task that has arrived locally and takes action to guarantee it. If the task cannot be guaranteed, the node can request for bids and can determine if some other node has the surplus to guarantee it. However, given that the task was sent to the best bidder and that the task's deadline will be closer than before, the chances of there being another node with sufficient surplus are small. Hence, we decided to send not only the task but also the identity of the second-best bidder to the best bidder. If the best bidder cannot guarantee the ask, it sends the task to the second-best bidder, if any. Otherwise, the task is rejected.

The environment that submitted the task will be responsible for appropriate action if a task is not guaranteed. One such action could be to resubmit the task with a later deadline. If there are specifications concerning, for example, the percentage of tasks that should be guaranteed, then the system should be designed to meet them.

We have been able to perform enough simulations to show that the algorithm is feasible. We are now in the process of making a number of extensions to the algorithms and trying to test them.

To summarize, our algorithm for scheduling tasks with hard real-time constraints in a loosely-coupled distributed system has the following main features.

1. It is a distributed scheduling technique; no node has greater importance, as far as scheduling is concerned, than any other.

2. The algorithm is designed to schedule both periodic tasks as well as nonperiodic tasks. The latter may arrive at any time.

3. A bidding approach is used for the selection of nodes on which tasks execute. We have worked out the details of the various phases involved in the bidding process.

4. Overheads involved in scheduling are taken into consideration. This includes the time spent on executing scheduling tasks at a node and the communication delays between nodes.

5. Heuristics are built into the algorithm. This is necessary given the computationally hard nature of the scheduling problem.

6. The algorithm can be extended to take into account other types of constraints, in particular, precedence and resource constraints.

7. Simulation studies are being performed in order to evaluate many of the alternatives available.

2.5 Stochastic Learning Automata

As part of this contract we developed a dcentralized scheduling algorithm based on a stochastic learning automata. For this to work, a number of extensions to the basic Stochastic Learning Automata were necessary. In this section we briefly describe what a stochastic learning automata is, our extensions to it, and some of the more interesting simulation results. The enhancements were necessary in order to deal with a number of issues, including how to keep the number of states of the SLA small, and how to deal with the interaction of multiple distributed SLAs and the associated problems such as stability.

A Stochastic Learning Automaton [NARE74] can be regarded as a sextuple $\{x,F,a,p,A,G\}$ where:

1. x is the input set. This set consists of allowable responses of the environment to the automaton. Responses can be one of several types, but here we only consider the binary response, i.e., {0,1}.
2. $F = \{f_1, f_2, \ldots\ldots, f_p\}$ is the set of internal states.
3. $a = \{a_1, a_2, \ldots\ldots, a_m\}$ is the action set. The automaton selects one of these actions,
4. $p(n) = \{p_1(n), p_2(n), \ldots\ldots, p_m(n)\}$ is the set of probabilities at time n, where n varies in discrete steps,
5. A is the algorithm which updates the probability set and is called the reinforcement strategy, and
6. G is the output function which maps the internal state into a corresponding action.

The environment in which the SLA operates evaluates each action of the SLA based on a probability vector called the penalty probabilities, $C=(c_1,c_2,\ldots,c_m)$. There exists one component of this vector for every possible action of the SLA. The action is deemed to be "bad" with the probability associated with that action. If these probabilities are static over time, the environment is called "stationary", otherwise it is called "non-stationary" which includes periodic, switched and slowly varying environments. Of course, the SLA has no a priori knowledge about these penalty probabilities. If one knew these penalty probabilities beforehand, it would be trivial to design a controller that minimizes the penalty received from the environment, insuring optimal behavior.

A Stochastic Learning Automaton operates as follows. The automaton initially selects a certain action from its action set. This action is chosen probabilistically according to an initial probability vector $p(0) = \{p_1(0),p_2(0),\ldots,p_m(0)\}$. This action generates a response from the environment. The response serves as a measure of the "goodness" of the particular action, or in our case just good or bad. The learning behavior of the automaton occurs in the following manner: if the response to a certain action at time n is favorable (non-penalty), increase the probability of selecting that action at time n+1, otherwise decrease it. This is called Reward-Penalty reinforcement scheme, where the reinforcement could be a linear or a non-linear function of the action probabilities. There are other schemes of updating the probabilities such as: Reward-Inaction (also called Benevolent Automaton), Inaction-Penalty, Penalty-Penalty, etc. The schemes used for reinforcement affect the rate of convergence of the SLA to

the optimal action and may cause instabilities by having the SLA alternate between actions or converge to a wrong action.

The main advantages of using a Stochastic Learning Automaton are:

1. The history of the responses from the environment are very simply modeled in the probability vector,
2. the reinforcement scheme can be tuned to improve the performance of the automaton and avoid instabilities, and
3. on each decision that the automaton makes, there are no iterative, highly time consuming calculations to be carried out. This makes it especially advantageous for real time control of a system.

In our approach, each host of the distributed system contains a scheduler, where each is modeled as an SLA. Although the SLA model has great potential, we cannot simply use it because the interactions among the set of SLAs is not defined in the SLA model. We have, therefore, developed extensions to the SLA model including:

1. A Measure of Goodness,
2. Network-wide States of Nature,
3. A Matrix Learning Automata, and
4. A Priority Encoded Associative Memory

1. Measure of Goodness "G": Every machine in the network is identified with a number of processes (in wait queue + execution) which to a reasonable degree of accuracy, is the optimal number of processes for that machine. If, for example, host A has its optimal number equal to 3, then, if it has less than 3 processes, it is considered to be underloaded; if it has greater than 3 processes

it is overloaded, and is a possible source of tasks to balance the network if needed.

2. Network-wide States of Nature: In performing distributed scheduling it must be possible for each host in the network to recognize underloaded hosts. Also, it must be possible to identify each network state. Thus, for N hosts, we would have 2^N possible states. This makes excessive demands on storage. We have consequently devised a heuristic to reduce the number of states, to provide for quicker balancing in the network, and to address stability.

   If, for instance, hosts A,B,C,D,E are underloaded and F,G,H,I,J are overloaded, then different overloaded hosts from the set F,G,H,I and J will be "a priori" assigned to recognize different underloaded hosts. Furthermore, at any given point, overloaded hosts recognize, at most, two underloaded hosts. This drastically reduces the possible states of nature. For example, for a network of 10 hosts, complete enumeration results in the need for 1024 states whereas our scheme requires only 55 states. In our scheme the states of nature would be as follows:

Host A underloaded

Host B underloaded

. .

. .

. .

. .

. .

Host J underloaded

which gives 10 states;

Host A,B underloaded

Host A,C underloaded

. .

. .

. .

. .

Host I,J underloaded

which gives an additional 45 states, making a total of 55.

3. Matrix Learning Automaton: As explained earlier, an SLA has a probability vector P(n) with which it selects possible actions. We have added the concept of Network-wide States of Nature to our model so that there is one probability vector P(n) for each network-wide state. Thus, the vectors of actions and probabilities of a basic SLA now becomes a matrix, one row of a matrix for each Network-wide State of Nature. The SLA must recognize the state, and it is aided in this task by a Priority Encoded Associative Memory.

4. Priority Encoded Associative Memory: A Priority Encoded Associative Memory is used to interact with the automaton and force it to work within the context of the state. There is one Associative Memory at each host. It works as follows: based on the state information passed to each host through the subnet, the Associative Memory outputs a pattern to the automaton which is, in fact, the row index into the Action and Probability matrices of the automaton. This pattern is generated by the associative memory and is equivalent to the state in which the automaton must operate.

As an example, for a certain host F, there might exist a priority sequence such as "ABCEDIGH". If, for instance, the network state observed by host F had hosts A,B,C,D,E, underloaded, the Associative Memory produces a pattern for the automaton which is an integer representing the row index into the two matrices and is equivalent to the state of nature "A,B underloaded". This means that if hosts A,B are underloaded, host F will ignore the other underloaded ones and will concentrate on balancing only a subset of the network. If however, A,B were not underloaded, but D,E,G, and I were, host F would recognize E,D as being underloaded.

For stability considerations, ordering can be easily varied from host to host by using a cyclic chain for generating a "start host", e.g., the order at host A might be BC,BD,BE,CD,CE...etc., and the order at host B might be CD,CE,CA,DE,DB....etc.

This means that if all hosts happened to recognize the same network state they would still react differently, i.e., they would be in different local states with different probability vectors.

Further, even if a number of hosts are in the same local states the reaction is probabilistic and not deterministic. This also lowers the chance of too much work being sent to one host.

Based on the above background we can now outline the operation of our algorithm. Each host periodically, with period x, checks its own status (underloaded or overloaded) and broadcasts this status to all other hosts. Using this state information, the PEAM at each host is able to determine the network-wide state which this local host is observing at this time. This state uniquely determines the probability vector to use in transmitting tasks for this interval of time. In parallel, and with period y, the scheduling algorithm also checks to see if the host is overloaded, and if so, it will transmit only 1 task to other hosts using the probability vector currently in effect, i.e., the destination host is decided probabilistically. Upon arrival at the destination, the receiving host decides to reward or penalize this action depending on the effect that the arrival of this new task has on its own state. The indiction of reward or penalty is transmitted back to the sending host which updates the probability vector in effect at the time of transmission of the task (and not necessarily in effect now). Each host in the network is operating in this manner, in parallel, and after a transition period the network "learns" the best actions for each of the network-wide states.

Detailed simulation results were reported in an earlier report. Here we summarize the main conclusions. One main conclusion of the simulation results is that using the extended SLA we designed, you can get good performance for small networks even if the number of states of the SLA is

relatively small. Keeping the number of states small makes use of this approach realistic, at least in this regard. Further, using too few states to describe the system results in poor performance and instabilities. Delays in large networks results in the information about the state of the network proving useless in getting the SLA to learn. To overcome this problem, methods for partitioning the network into subsets of cooperating nodes might be feasible, but this was not investigated further. Again, for small networks we found that the extended SLA scheduling algorithm handled burst of job arrivals very well. We were also able to show that the extended SLA algorithm performed better than a number of baselines and analytical models. The main problem with the SLA approach is that it is restricted to simple job scheduling and does not seem easily extensible to more complicated process scheduling, e.g., where processes are going to cluster or have special resource requirements.

3.0 Conclusions and Recommendations

Under this contract we developed and analyzed 7 decentralized controlled scheduling algorithms by implementing highly parameterized simulation programs for each. The algorithms varied from simple job scheduling algorithms, to complicated process scheduling algorithms, to algorithms that operate in an environment where tasks have real-time constraints such as deadlines. Some of the algorithms we developed used explicit feedback and some did not. Overall, many novel ideas were generated and shown to be effective. In total, eleven refereed papers and two master's theses were produced as a result of our work. The main conclusion of each aspect of all this work is summarized in this final report. All papers and theses have been submitted earlier, as well as 11 quarterly reports.

While much was accomplished, there remains many interesting problems. Even our most complicated algorithm (bidding based on a McColloch Pitt Neuron) makes many assumptions. Removing assumptions and dveloping algorithms to handle the increasing complexity is important. For example, one can remove the assumption that all processes of a cluster have equal attraction for one another, or remove the assumption that processes in execution require all objects that it needs to be local (i.e., allow remote access). It would be interesting to see how how results change by adding more complications.

There are other issues that we barely touched on and these need to be investigated. They include stability and robustness issues for decentralized control algorithms. At a minimum, we recommend continued efforts in the stability and robustness areas.

4.0 Budget

	Planned	Spent
Year 1 and 2	69,783.83	69,783.83
Year 3	61,046.67	61,000.00
Planned	130,830.50	130,783.83

) List of Publications (J. Stankovic)

"Scheduling Tasks With Resource Requirement in Hard Real-Time Systems," with Wei Zhao and Krithivasan Ramamritham, submitted to IEEE Transactions on Software Engineering, May 1984.

"An Adaptive Bidding Algorithm For Processes, Clusters and Distributed Groups," with Inderjit Sidhu, Proceedings 4th International Conference on Distributed Computing Systems, May 1984.

"Dynamic Task Scheduling in Distributed Hard Real-Time Systems," with Krithivasan Ramamritham, IEEE Software, Vol. 1, No. 3, July 1984.

"An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling," IEEE Transaction on Computers, February 1985.

"Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," Computer Networks, Vol. 8, No. 3., pp. 199-217, June 1984.

"A Heuristic For Cooperation Among Decentralized Controllers," Proceedings INFOCOM 83, invited paper, April 1983.

"Achievable Decentralized Control For Functions of a Distributed Processing Operating System," Proceedings of COMPSAC, November 1982.

"An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of Decentralized Control Algorithms," with Noor Chowdhury, Ravi Mirchandaney, and Inderjit Sidhu, Proceedings of COMPSAC, November 1982.

"Software Communication Mechanisms: Procedure Calls Versus Messages," IEEE Computer, Vol. 15, No. 4, pp. 19-25, April 1982.

. "The Analysis of a Decentralized Control Algorithm for Job Scheduling Utilizing Bayesian Decision Theory," Proceedings 1981 International Conference on Parallel Processing, pp. 333-340, August 1981.

. "A Comprehensive Framework for Evaluating Decentralized Control," Proceedings 1980 International conference on Parallel Processing, pp. 181-187, August 1980.

6.0 Master's Theses Produced

1. Ravi Mirchandaney - Decentralized Job Scheduling Using a Network of Stochastic Learning Automata
2. Inderjit Sidhu - Decentralized Process Scheduling in a Distributed Environment

7.0 Distribution List

Defense Technical Information Center — 2 copies
Cameron Station
Alexandria, VA 22314

Commander — 2 copies
USAERADCOM
ATTN: DELSD-L-S
Fort Monmouth, NJ 07703

Commander — 1 copy
USACECOM
ATTN: DRSEL-COM-RF (Dr. Klein)
Fort Monmouth, NJ 07703

Commander — 1 copy
USACECOM
ATTN: DRSEL-COM-D (E. Famolari)
Fort Monmouth, NJ 07703

Commander — 10 copies
USACECOM
ATTN: DRSEL-COM-RF-2 (C. Graff)
Fort Monmouth, NJ 07703

END

FILMED

9-85

DTIC